特集号
招待論文

NetCOBOLのHadoop連携機能の開発と実践事例

上田 晴康[†1]　榎田 勉[†2]　立岩 恭也[†2]

[†1]（株）富士通研究所　[†2]富士通（株）

基幹バッチをHadoop上で実行することで高速処理をしたいという要望が多くなっている．基幹バッチに関しては，既存のCOBOLアプリケーションとデータ資産が大量にあるため，これらを修正・変換することなく利用できることが重要である．しかし，複数入力ファイルを用いる突合せを伴うアプリケーションはHadoopでの実行が困難なため，新規開発が必要になる事が多い．これを解決するためHadoopマルチプレクサ技術を開発した．また，HadoopからCOBOL特有のデータフォーマットを扱うためのCOBOLデータ対応技術も開発した．この技術を業務へ実適用して，アプリケーションの修正なく，並列実行することによる高速化の効果があることを示した．

1. はじめに

昨今，お客様ビジネスにおける競合他社との差異化や，企業や業種を超えた新たなサービスのために，いかにビッグデータを活用していくかが，大きな課題となっている．

ビッグデータ活用の本質は，ビジネス活動で発生する様々なログや情報から新たな価値を発見し，また様々なデータの相関関係から新たなトレンドや予兆を発見し，ビジネスの変革，創出につなげていくことだと弊社は考えている．

一方，弊社のお客様におけるビッグデータ利活用の目的（**図1**）を見てみると，新たなビジネス領域や埋もれているデータの利活用のビジネスだけでなく，既存のビジネスの範囲を含めた様々なシーンで，ビッグデータ活用をしたい，というニーズが見えてきている．

特に多いのが，47％を占める「既存領域のデータ処理改革」のニーズである．具体的には，既存のバッチ処理を高速化するために，ビッグデータ技術を活用したいというニーズである．

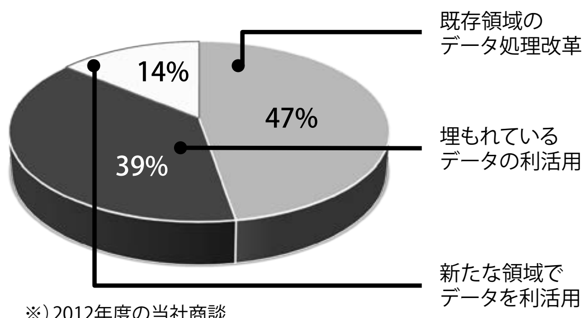


図1　ビッグデータ利活用の目的

次に多いのは39％を占める「埋もれているデータの利活用」である．企業の内外に実は存在する，使っていなかった様々なデータを組み合わせて分析をすることで，従来気づかなかった事象や法則を抽出し，サービスの付加価値や業務の精度向上を狙うものである．

「新たな領域でのデータ利活用」は，ビッグデータを活用した新たなビジネスの創出を指すが，これは，未だ14％にとどまっている．

本稿では，既存のバッチ処理の高速化のニーズに答え，メインフレーム時代からそのまま多く利用され続けているCOBOLのバッチ処理アプリケーションを，OSS(Open Source Software)のApache Hadoop[1]で並列分散処理しバッチ処理時間を短縮する，弊社のCOBOL開発・運用ソフトウェア「FUJITSU Software NetCOBOL」（以降，NetCOBOL）[2]のHadoop連携機能の開発と実践事例について述べる．

2. バッチ処理について

本章では，バッチ処理の遅延が及ぼす問題と，バッチ処理時間の短縮の課題について述べたあと，バッチ処理の基本パターンを説明する．

2.1　バッチ処理時間の遅延による問題

基幹系の業務システムは，昼間のオンライン処理で業務データを溜め込み，その溜め込まれた大量の業務デー

 2014 ©Information Processing Society of Japan

タを夜間のバッチ処理で集計・更新・加工する構成が多い．

たとえば，POSシステムによって更新された売り上げデータを集計したり，納品データから在庫情報を更新したりする処理が挙げられる．

今日，データ量の増加により，バッチ処理のスピードが追い付かず，計画通りの時間に完了しない（突き抜ける）事態が生じ，問題になっている．

金融機関でも，夜間バッチ処理での送金処理が完了せず，翌日のATMサービスの開始時間の遅れ，また，突き抜けによりオンラインが開始できず, 保険金の支払い・返金や契約申し込みができなくなった，などの問題がおきている．

このように，突き抜けが発生すると，社会的な問題に繋がる場合もあるため，バッチ処理は既定の時間までに必ず終了することが重要である．

2.2　バッチ処理時間の短縮の課題

バッチ処理時間を短縮するため, 多くのシステムでは, バッチ用サーバのスケールアップと手組み（既存の製品や雛形などを流用せず自前で開発すること）による処理の並列化の対策が取られている．しかし，これらの対策は以下の理由から大きなコストがかかったり，拡張性に課題がある．

- **バッチサーバのスケールアップ**
  サーバのハードウェア拡張は上限があるため，スケールアップには限界がある．また，スケールアップ可能なハイパフォーマンスコンピュータは，一般的に高価である．
- **手組みによる処理の並列化**
  アプリケーションを並列処理するための制御を自前で行う必要がある．また，並列処理のためには各タスクに割り当てる処理データの分割を事前に行う必要がある．このため，データ量が増加し，並列度を上げる際には，タスク制御の変更，データの分割や割り当てが必要になる．

2.3　バッチ処理への Hadoop 適用の期待

Apache HadoopはOSSであり，近年注目されている技術である．Hadoopが備える以下の特徴は，バッチ処理の課題を解決できるものであり，Hadoopのバッチ適用が注目されている．

- **バッチサーバのスケールアウト**
  並列分散処理の基盤を備えているため，高価なハイパフォーマンスコンピュータを使うことなく，低コストの汎用PCサーバで実装できること．また，データ量の増加に対し，サーバを追加するだけで処理能力を上げられるため，スモールスタートが可能であること．
- **並列分散処理の自動制御**
  入力データは自動分割され，各サーバのタスクに割り振られるため，アプリケーションの並列制御は不要であること．また，Hadoopの構成ファイルの設定だけで多重度，タスクのリトライ数，タスクのタイムアウト時間などの変更ができるため，システム構築が容易なこと．

このことから多くのHadoop専門のベンチャー企業や大手ITベンダーがHadoopを含むディストリビューションを提供している．例えばCloudera社[3]，MapR Technologies社[4] Hortonworks社[5]，IBM[6]，富士通[7]，などである．

また，基幹系の業務システムへのHadoopの適用も始まっている[8].

2.4　バッチ処理の高速化に期待される要件

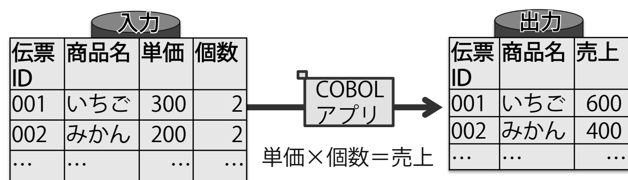
COBOLのバッチ処理を分析すると，**図2**の3パターンの組合せでバッチ処理が実現されていることが多い．

このバッチ処理の基本パターンは，Hadoopの処理の流れと相性がよい．このため，バッチ処理へのHadoopの適用による高速化が期待されているが，同時に求めら

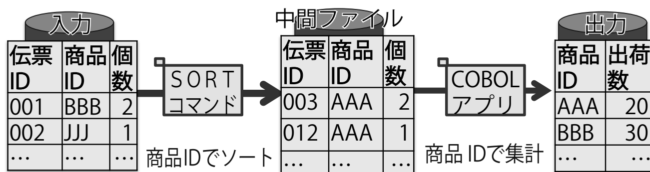
COLUMN　Hadoop の動作概要

Hadoop は，大規模データの分散処理をするためのフレームワークである．耐故障性と並列処理の性能を出すために，以下の仕組みを持っている．

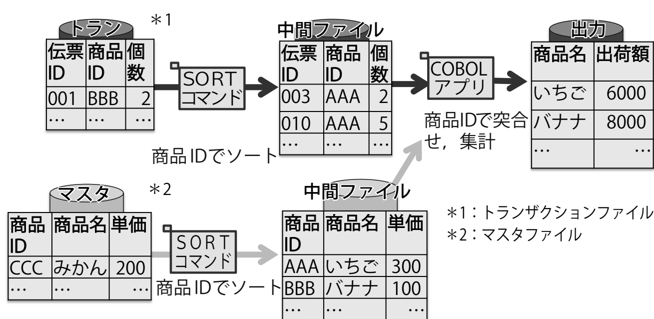
- Map: Hadoopの分散ファイルシステムにおさめられたデータを読み出し，集約するためのキーの抽出処理をするための仕組みである．Mapでの処理方法をJavaのクラス名で指定することで，分散配置されたデータを並列実行する．データ読み込み時に適切に入力レコードを取り出すためにInputFormatと呼ばれるクラスを利用する．
- シャッフル・ソート: Hadoopの中核部分であり，Mapの出力をネットワーク経由で集約するために，並列ソートの技法を用いている．
- Reduce: Mapの出力をキーごとに集約した結果を元に，出力を行うための仕組みである．Map同様Javaのクラス名を指定する．



(a) レコードの単純加工：
すべてのレコードに同じ処理を行う



(b) 特定のキーを持つレコードの集計：
同一キーのレコード単位に処理を行う



(c) ファイル同士の突合せ：
複数ファイルの突き合わせ処理を行う

図 2　バッチ処理の基本パターン

れる要件を以下に挙げる．

- **既存COBOLアプリケーションの流用**
  基幹系処理で動作実績のあるアプリケーションを流用することで，高品質なシステムを短期構築することが可能になる．逆に，アプリケーションの修正が必要な場合には，影響範囲の分析，修正方法の検討，単体テスト，結合テストを行うため，コストがかかるとともに修正ミスのリスクを伴うデメリットがある．このため，既存アプリケーションの修正を最小限に抑え，動作できることが求められる．
- **既存データの流用**
  既存データを流用できない場合，利用可能なフォーマットに合わせて変換が必要である．しかし，データを変換をした場合，その変換後のデータに合わせてアプリケーションの修正が必要になる．また，フォーマットによってはデータ変換前と後とで同じデータを表現できないケースがある（バイナリデータとテキストデータの変換など）．

さらに，対象のデータ量が大きい場合，変換を行うとその時間がかかってしまう．これらの理由から，既存データをそのまま利用できることが求められる．

2.5　Hadoop および関連技術とその課題

2.5.1　既存アプリをHadoopで実行するための課題

Hadoopで既存のアプリケーションを実行するための技術として，Hadoop Streamingがある．これを用いることで並列に実行されるMapおよびReduceのそれぞれの処理内容として,既存のアプリケーションを指定できる．MapおよびReduceで処理すべきデータは，標準入出力経由で受け渡される．

しかしながら，Hadoop Streamingでは，既存のアプリケーションとのデータのやり取りが標準入出力に限定されているため，複数ファイル入出力という用途に利用することができない．

2.5.2　既存システムとのデータ連携

HadoopのデフォルトのデータフォーマットはCSV(comma-separated values)やTSV(tab-separated values)を含むプレーンテキストであるため，より複雑なデータ構造を持つレコードの扱いには工夫が必要である．多くの場合，XML，JSON，Apache avro[9]，Apache thrift[10]など改行文字のエスケープや，データ構造のテキスト文字への変換が定義された技術が用いられる．

一方，基幹バッチにおいて使われるCOBOLに関しては，これらでは解決できない．なぜならばCOBOLでは規格[11]で定められたデータフォーマットを用いる必要があるためである．このデータフォーマットは，COBOLアプリケーションで定義されたデータ構造に従って整形され，10進数値や固定長文字列などがセパレータなしに連続している．

Hadoopには一般的なCOBOL向けのInputFormatはこれまでなかったため，既存システムとデータ連携するには，CSVやTSVファイルを出力し，それを処理する必要があったが，2.4節で述べたとおり既存データの流用の観点から望ましくない．

2.5.3　複数入力の処理

Hadoopは並列ソートを行うシャッフル・ソートを技術的なコアとして持つため，2つ以上の異なるファイルを扱うのは得意ではない．すなわち，2.4節のファイル同士の突合せで示したパターンを処理するのは困難である．これに対し，いくつかの限定された対処法がある．

Hadoopでは複数入力の突合せ処理をするのが難しい

(1)　ジョブキャッシュの利用

複数入力のうちの一つだけをHadoopで処理するデータとし，残りの入力はジョブキャッシュと呼ばれる領域でアプリケーションに渡す方法がある．この場合，アプリケーションは，MapやReduceタスクでデータを処理するのに先立って，残りの入力ファイルを読み込んでメモリ中に保持することになる．

一つのデータだけが巨大な場合にはこの方法でもよいが，複数のファイルのいずれが大きいかをあらかじめ想定してアプリケーションを開発する必要がある．このため，汎用性に劣り，またデータ量が大きく変動した場合に対処できないという問題がある．

(2)　MultipleInputFormat

MultipleInputFormatはHadoopのデータ読み込み手段を拡張するためのものでHadoopに標準で含まれている．これを用いることで，入力ファイルごとに異なるInput-FormatとMap処理を指定することができる．

しかし，Hadoopの利用法としては原始的な方法であり，MultipleInputFormatの利用を定義するためのアプリケーションと個々のファイルフォーマット用のMapアプリケーションをそれぞれ新規に開発する必要がある．このため,性能は良いものの多くの開発工数を必要とし，非常にデータ量が多いために性能を重視する場合以外は適用が困難である．

(3)　Apache Hive

Apache Hive[12]は，CSVやTSV形式のHDFS上のファイルをRDBのテーブルとみなして処理するための処理系である.複数入力を処理するもっとも重要なユースケースはRDBでもしばしば行われる突合せ処理（Join）であるため，SQLとほぼ同じクエリ言語であるHiveQLを用いることで複数入力の処理の多くを記述することができる．

しかし，COBOLのデータフォーマットに対応していないために，データ連携の観点から適用は困難である．

(4)Asakusa Framework

Asakusa Framework[13]は突合せ処理も含む処理一般をHadoop向けに開発するための開発環境である．asakusaではJAVAに独自の記法を追加した言語を用いて，ジョブネットから個々のジョブまでを記述することができ，この記述を元にHadoopアプリケーションの自動生成を行う．

しかし，asakusaを用いる場合は，ジョブネットから扱うデータ構造の定義を含む個々のジョブのアプリケーションまでの全体を通した新規開発が必要である．このため，既存のCOBOLバッチを流用することはできない．

3.　Hadoopマルチプレクサおよび COBOLデータ対応技術

前章で示したように，Hadoopで実行する際は，ジョブの中のソート・突合せというパターンを実行するのが一番難しい．そこで，このパターンのCOBOLアプリケーションを無修正のまま用いて処理する技術を開発した（図3）．

このような動作を行うため，NetCOBOLでは下記の機能を開発した．

1．汎用のreduce side join
2．Hadoopマルチプレクサ技術
3．COBOLファイル対応技術

以下の各節でこれらの機能について概説する．

3.1　汎用のreduce side join

2.5.3項で述べたとおり，Hadoopで複数の入力ファイルを扱うのは難しい．そこで，突合せ（Join)に特化して複数の入力ファイルを処理するための機能を提供する．

HadoopにおいてJoinを行う方式としては，In Memory Join，Map Side Join，Reduce Side Joinなどの方式がある[14].

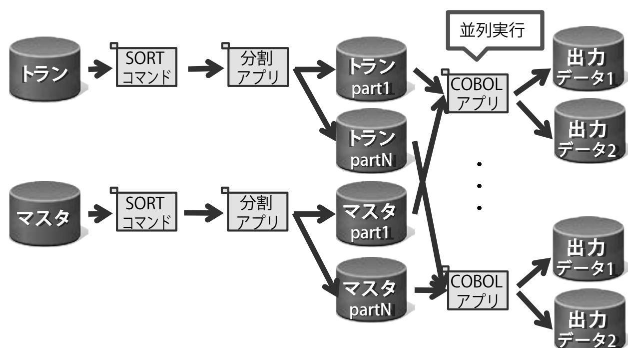
(1)　In Memory Join

In Memory Joinは，ジョブキャッシュファイルを用いてMapタスクとしてJoinを行う方法である．Joinする対象ファイルのうち，一つだけをMapタスクの入力ファイルに指定し，それ以外のファイルはあらかじめメモリに読み込んでおいてJoinを実行する．2.5.3項で述べたとおりジョブキャッシュファイルを用いる方法にはデータ量が大きく変動すると対応できないという問題があるため採用しない．

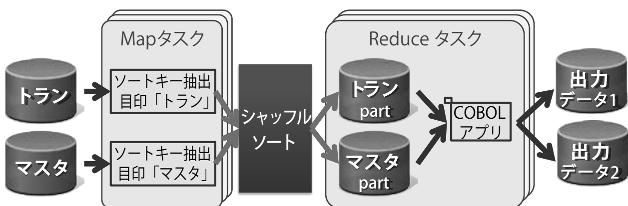
(2)　Map Side Join

Map Side Joinは，MapタスクとしてJoinを行う別の方法である．複数のファイルをあらかじめ，同じキーの範囲を持つパーティションに分けておき，同じパーティションのための複数のファイルをMapタスクで読み込みJoin処理をする．

この方式は，パーティションファイルがきちんとスレーブサーバに配置されていないと処理効率が落ちてしまうという問題がある．そして，パーティションファイルをきちんとスレーブサーバに保存するためにはHadoopのジョブを別に実行させる必要がある．これは次に述べるReduce Side Joinの前半の作業と等価であり，パーティションに分けたファイルを何度も処理する場合でなければわざわざ別のHadoopジョブに分けて実行するメリットはない．



(a) 突合せ処理の従来の並列実行



(b) 突合せ処理の Hadoop における実行

図 3　Hadoop マルチプレクサの概要

(3)　Reduce Side Join

Reduce Side Joinは，MultipleInputFormatを用いて複数の入力フォーマットのファイルを読み込み，それぞれのフォーマットからソートキーの値を抽出してHadoopのシャッフル・ソートを行う（図4）．この際に，何番目の入力フォーマットから来たデータであるかの目印を各レコード（Key-Valueのペア）に付けておく．これにより，Reduceでは，同じキーにJoinされた複数の入力フォーマットのデータを受け取ることができる．Reduceでは，何番目のファイルフォーマットから来たデータかを判別した上で，それらの情報を組み合わせてJoinを行うことができる．

この方式は，二つ以上のデータフォーマットのデータの量がどちらも多い時に使える汎用的な方法である．そこで，この方式を設定ファイルのみで実行できるようにした．

3.2　Hadoop マルチプレクサ技術

Hadoopマルチプレクサは，Hadoop Streaming同様，Hadoopのタスクとして一般のアプリケーションを起動して，起動されたプロセスとタスクがデータ交換できるようにする．Hadoop Streamingと異なり，起動するアプリケーションは引数等で指定されたファイルを読み書きするものとする．

汎用のreduce side join機能から得られたデータは，何番目のファイルフォーマットから来たデータか目印を用いて判別された後，元と同じフォーマットの複数のデータに整形される．このデータをディスク上のファイルとして書き出してからアプリケーションを起動することも

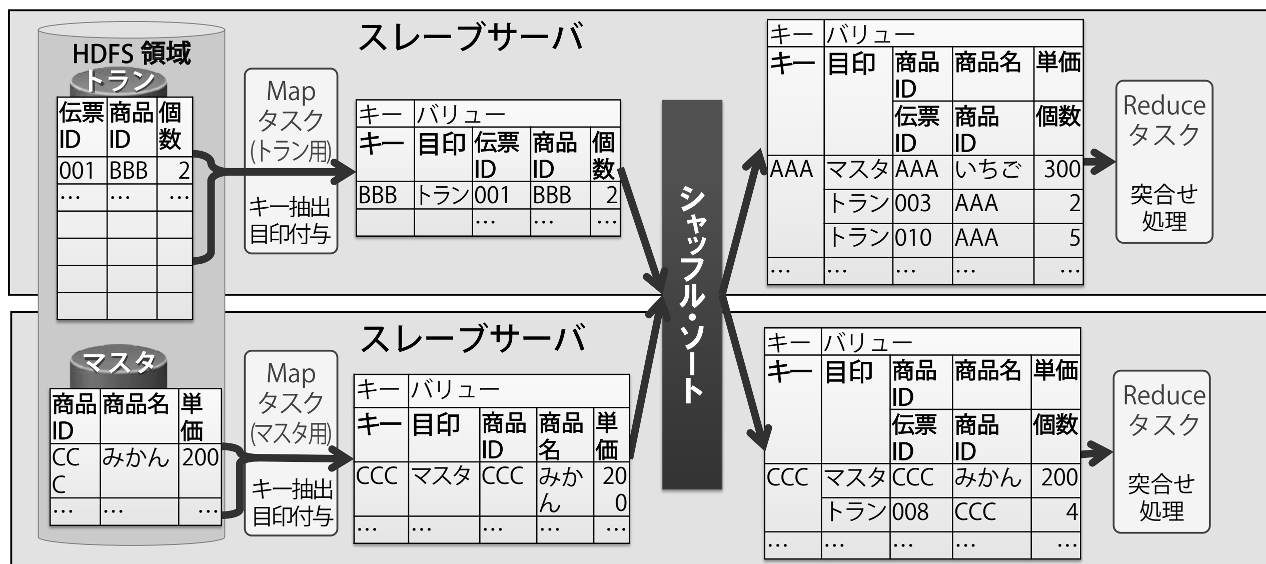


図 4　Reduce Side Join

できるが，その場合は，ディスクへの総読み書きデータ量が入力データ量と同じだけ増えてしまうため処理時間が増大する．また，起動されたプロセスはReduceタスクによるディスク読み込みと並行動作可能なものであるが，逐次動作となりこれも処理時間増大の要因となる．

そこで，Hadoopのタスクと起動されたプロセスが名前付きパイプを用いてデータ交換する仕組みとした（**図5**）．名前付きパイプは，Linux等のOSに広く存在するプロセス間通信の仕組みである．通常のファイルの読み書きと同様のインタフェースを持ちながら，複数のプロセスがパイプと同様の通信ができる．

しかし，パイプとしての性質があるため，パイプのバッファ領域以上の読み書きにおいては，名前付きパイプは書き込みプロセスと読み込みプロセスは同期して動作しなくてはならないという制約がある．すなわち，名前付きパイプへの書き込みを行うと，その名前付きパイプからの読み込みが終わるまでブロックされる．パイプのバッファ領域は最近のLinuxでは64KBであり，ビッグデータの読み書きを行う際にはほぼ常に同期すると考えてよい．

一方，一般のアプリケーションは，複数の入力ファイルをどのような順序で読み込むかはわからないことが多い．Reduceタスクに届く複数の入力ファイルフォーマットのデータの書き込み順序が一般のアプリケーションの読み込み順序と異なるとブロックされてしまうのでは，容易にデッドロックが起きてしまう．

そこで，Hadoopマルチプレクサは，汎用のreduce side join機能から届いたデータを直接名前付きパイプに書き出すのではなく，メモリ中のバッファを介し，名前付きパイプごとに用意されたスレッドを用いて名前付きパイプに書き出す．

Hadoopマルチプレクサは，上記のようにReduceタスクにおける複数入力をするアプリケーションに対応する．さらに，各入力ファイルフォーマットに対する抽出処理を一般のアプリケーションで行う用途のために，Mapタスクにおける１入力１出力のアプリケーションの起動にも対応する．ただしHadoop Streamingと異なり，データ交換には標準入出力を使わず，名前付きパイプを使う．

なお，一般のアプリケーションがそれぞれの入力ファイルを最後まで読んでから次の入力ファイルを読みだすというような想定外の動作をした場合にメモリ中のバッファがあふれることがあるため，回避策としてディスク上の領域をバッファとして利用する機能も持っているが，本稿では詳述しない．

3.3　COBOLファイル対応技術

COBOLには，シーケンシャルに読み込み可能なファイルのフォーマットとして，以下の２種類がある．

- **レコード順ファイル**
  データを順次蓄積する場合や，大量データを保存する場合に用いられるファイルであり，レコード長には固定長と可変長がある．
- **行順ファイル**
  改行文字をレコードの区切りとするファイルであり，レコード長は先頭から改行までの長さとなる．

前述のとおり，これらのCOBOLファイルはHadoopで直接読み書きすることはサポートされていない．このため，COBOLファイルとデータの入出力に対応した独自クラスの実装を行うことが必要である．

また，シャッフル・ソートではデータのソート処理を行うが，デフォルトではバイナリ順でソート処理が行われる．COBOLデータは内部表現がバイナリ順ではないデータ形式があり，バイナリ順では正しい結果とならな

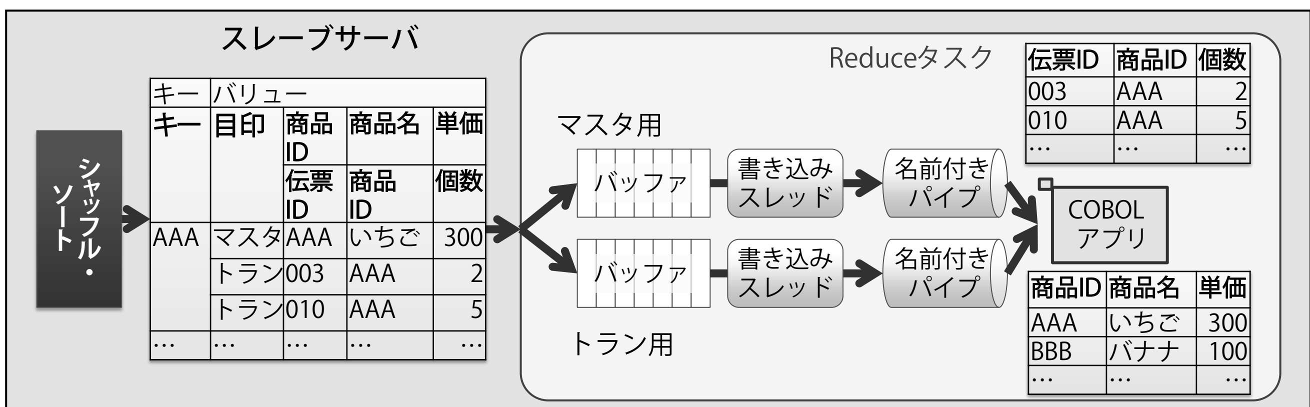


図5　Hadoopマルチプレクサ技術

**COBOLファイルの
Hadoopでの読み書きが
資産活用のポイント**

いため，ソート処理についても独自クラスの実装が必要である．COBOLデータに対応するために実装が必要なHadoopの基底クラスを以下に挙げる．

- InputFormat,OutputFormat
  COBOLファイルを読み書きする
- InputWriter,OutputReader
  HadoopとCOBOLアプリケーションとのデータをやり取りする
- WritableComparator
  COBOLデータについて正しいソート順を返す

これらのうち，本項では特に対応に苦慮したレコード順ファイルの可変長を読込む処理を詳しく述べる．Hadoopではファイルが既定のサイズによりブロック分割され，各Mapタスクの入力になる．このため，レコードが分断されてMapタスクに割り当てられる可能性があり，最初にレコードの先頭を正しく位置付けしなければ，誤ったデータを読み込んでしまう．レコードの先頭位置は，例えば行順ファイルの場合，改行文字を検索することで検出可能である（**図6**）．

しかし，レコード順ファイルはバイナリデータであり，改行文字のような検索可能なデータが存在しない．また，固定長であればオフセット値がレコード長の倍数であるかどうかを判定し位置づけることが可能であるが（**図7**），可変長の場合レコードごとに長さが異なるため，そのような位置付けは不可能である．

このため，本機能ではあらかじめ，レコード順ファイルをパースし，Hadoopによって分割される切れ目位置を調べておき，Mapタスクに割り当てられた先頭位置を補正することで先頭位置を位置づけることとした（**図8**）．

4. 業務への適用

4.1　適用対象の選定

Hadoopを適用する場合，まず，対象となる業務のジョブネットのどこで時間がかかっているか，全体を精査する．Hadoopではデータを自動的に分割し，各タスクに割り当て並列分処理が行われるため，レコード間の依存関係がなく多重で実行できる処理が対象になる．ただし，後述の理由から，データベースの更新をしているものは対象外となる．

次に，Hadoopに適用するジョブが決定したら，他のジョブは既存システムに残し，選定したジョブだけをHadoopに移行する．つまり，既存システムに，Hadoop用のサーバをアドインする，というシステム構成になる

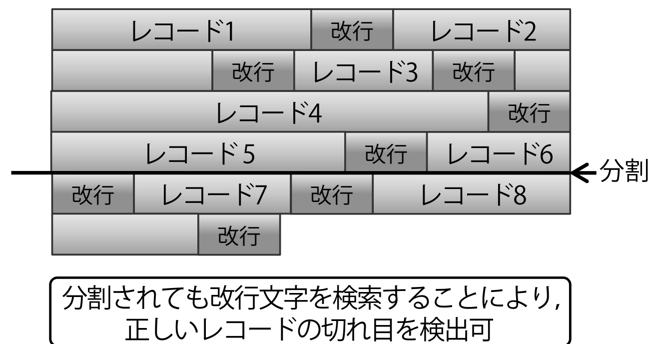


図6　行準ファイルのレコード検出

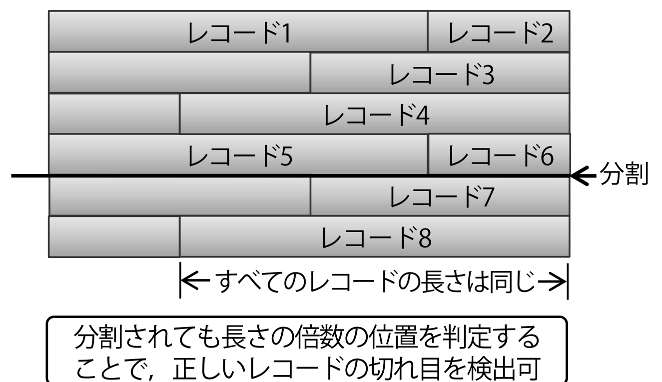


図7　レコード順ファイル（固定長）のレコード検出

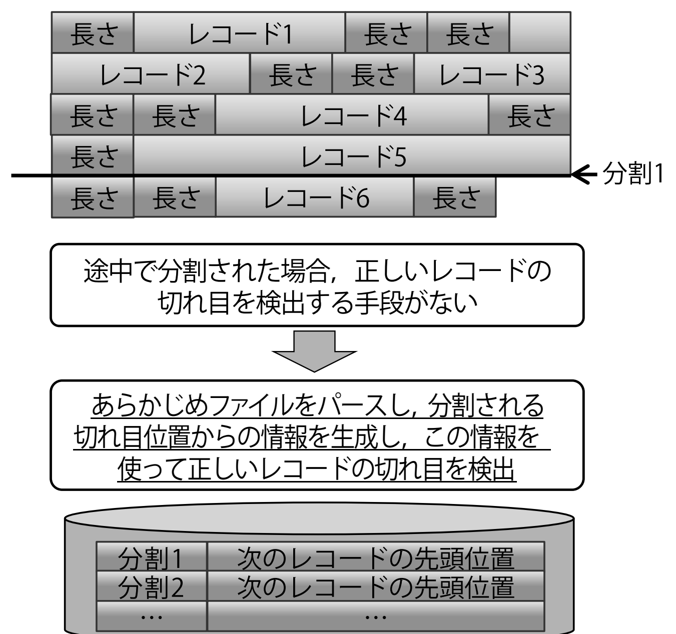


図8　レコード順ファイル（可変長）のレコード検出

ことが多い．こうして，バッチ処理のジョブネット全体の処理時間を短縮するのが，既存のバッチにHadoopを適用する場合の方法になる．

4.2　データベースの利用上の注意

業務で利用されているバッチ処理では，データベースを利用しているものも多いが，Hadoopでは，データベースの利用には注意が必要である．MapReduceアプリケーションは多重実行されるため，データベースサーバにアクセスが集中し，性能ネックになる場合があるためである．したがって，データベースを利用した処理はファイルを使用する処理へ変更し，必要に応じてジョブ終了後ファイルからデータベースへアップロードする方法が望ましい．

また，以下にデータベースの利用方法ごとの注意点を述べる．

- **データベースの参照**

データベースサーバの性能ネックが，ディスクIOのボトルネックに起因する場合，データベースの物理構造を分割することで解消できる場合がある．

具体的には，テーブルのパーティション化を行い，データを物理構造上分割することでレスポンスを向上させる．特にReduceタスクでは，キーごとにグループ化されたデータを処理し，他のReduceタスクとキーが重複しないことから，データのグループ化に使用される「キー」と，物理構造上分割する単位とを同じにしておくと効果的である．

- **データベースの更新**

Hadoopには以下の耐障害機構[15]が備えられている．

- タスクの障害による再実行
- 投機的実行
- TaskTrackerの障害によるタスクの再実行

これらにより，MapReduceアプリケーションからデータベースの更新処理を行うと，テーブルの不整合が生じる恐れがある．例えばジョブ失敗時，特定のタスクの結果のみがデータベースに反映された状態になる場合である．このため，MapReduceアプリケーションからデータベースの更新を行うべきではない．

4.3　データ転送

4.1で示したように，業務バッチの特定のジョブだけをHadoopに移行することになる．通常，バッチでは，先行のジョブの出力ファイルが後続のジョブの入力ファイルとなる．このため，Hadoopに移行したジョブの前後で，それらのファイルのHDFS(Hadoop Distributed File System)へのアップロードとHDFSからのダウンロードが必要になる．バッチの処理時間には，これらの時間をあらかじめ見込んでおくことが必要である．

なお，弊社の並列分散処理ソフトウェア「Interstage Big Data Parallel Processing Server」[7]を用いることで，独自技術によりHDFSへのアップロード/ダウンロードを不要とし，転送時間の削減が可能である．

高速化のポイントは
処理単位を細分化する
最適なキーの選択

4.4　技術検証

弊社では，この技術を実業務に適用してHadoop適用の効果がどの程度みられるか，技術検証を行った．それらのうち，Hadoop特有の課題が上がったものを以降で説明する．

検証対象は，メインフレーム上の既存バッチ処理であり，8多重実行し31分を要している処理を対象とした．対象の処理は，マスタファイルとトランザクションファイルとを突合せる処理であり，これらのファイルはレコード順ファイルの可変長であった（**図9**）．

このバッチ処理をHadoop上で実行し，1～15多重（スレーブサーバ1～3台）で実行した場合の処理時間を測定した．なお，本検証においてアプリケーションの流用率は100％であり，既存アプリケーションをそのまま活用できた．測定の結果，15多重時が15分と最短となり，メインフレームの約2倍のスループットを実現できた．実行結果を分析したところ，この業務ではCOBOLデータ内のキーに偏りが見られ，特定のReduceタスクにデ

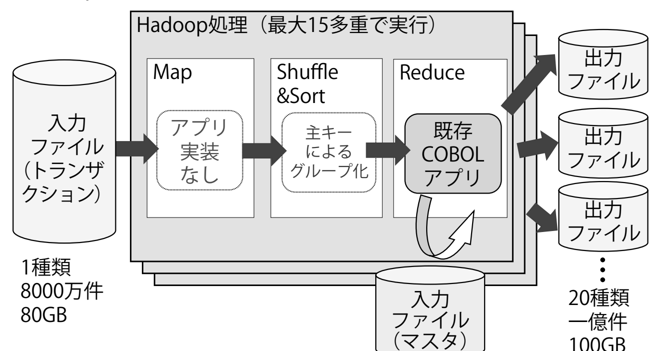


図9　技術検証で用いたシステムの構成

ータが集中しており，そのタスクの終了待ちになっていた（図10）．このため，これ以上多重度を上げても性能向上は見込めない．

キーの偏りを改善するため，キーの設定を細かくし，処理単位を細分化することで，さらなる性能向上が見込める（図11）．この場合，データのグループ単位が変更になるため，たとえば，集計処理を行っている場合は集計する単位が変わるため，その対応も行うことが必要である．

5.　実適用上のポイント

前節で述べた技術検証では，キーの偏りが見られたが，弊社のほかの事例でもキーの偏りが見られることがある．また，技術検証では入力ファイルが70GBであり，Hadoop適用の効果が見られたが，実業務バッチでは少ない入力データの処理も多い．

本節では，キーを分散する工夫と，Hadoop適用効果のある既存バッチの入力データ量について述べる．

- **キーを分散する工夫**

データをReduceタスクに割り振るルールは，キーからハッシュ関数によって求めたハッシュ値によって決定している．しかしながら，キーの値とReduceタスクの多重度によってはハッシュ値が重複し，結果的に偏りが発生する場合がある．この対処としては，データに合わせたハッシュ関数の改良や，データのサンプリングに基づく振り分けを行う方法があるが，キーの分布が明らかな場合，ハッシュ関数ではなく，Reduceタスクへの割り当てルールを規定することで，理想的なキーの分散を実現できる場合も多い．これを実現するために，シャッフル・ソートで，あらかじめ規定したルールに則ってReduceタスクへの振り分けを行う機能を実現した．

- **Hadoop適用効果のあるバッチの入力データ量**

Hadoopは並列分散処理を行うために，ジョブの起動停止や各ノード間の通信によるオーバヘッドがあり，これらに数十秒の時間を要する．このため，もともと短時間の処理をHadoopに適用しても時間短縮の効果は薄く，いかにデータ量が多く長時間の業務を適用させるかが重要である．

弊社ではデータ量とHadoopによる時間短縮の効果の関係を確認するため，ファイル同士の突合せパターンを用いて，従来のバッチ形式（1多重）とHadoop適用のバッチ形式（Reduce:1 ～ 4多重）の処理時間を測定した．この測定モデル（図12）では，データサイズが1 ～ 4GBのとき，従来のバッチ形式と比較して性能が劣化するか，大きな性能向上が見られなかったが（図13），

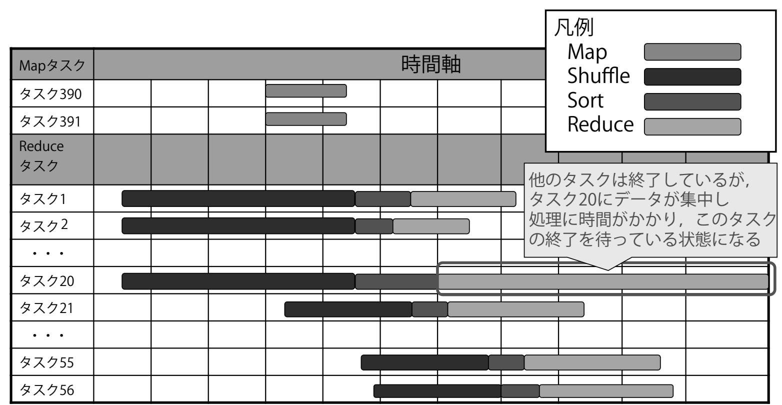


図 10　技術検証時のガントチャート

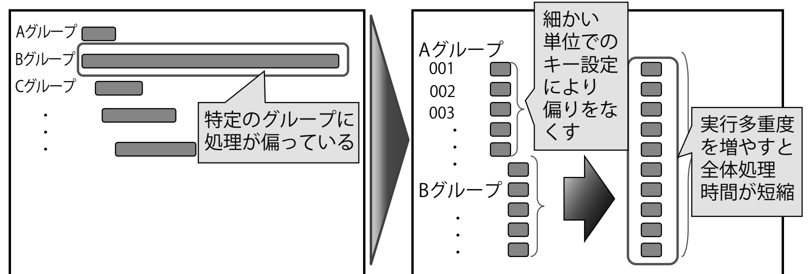
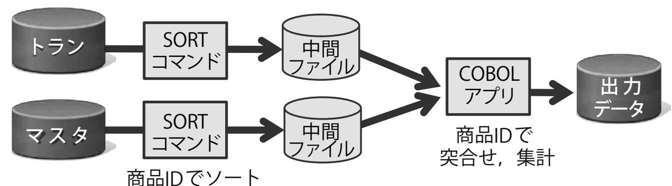


図 11　Reduce タスクのデータ量の偏りの改善による効果



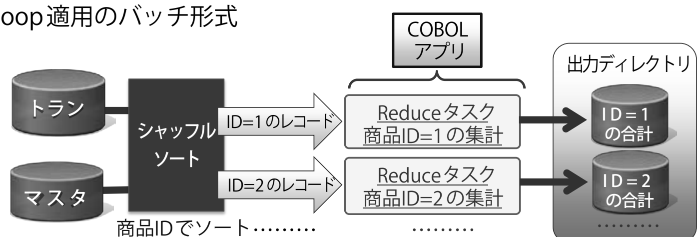


図 12　測定モデル

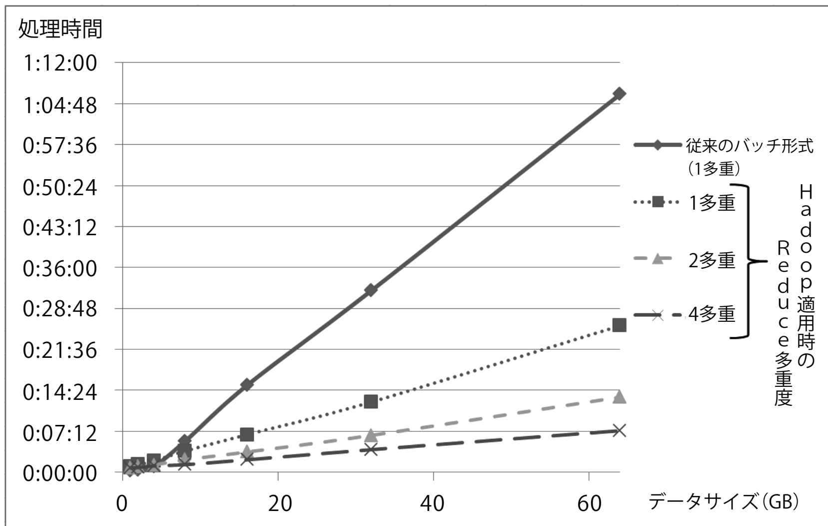


図 13　データ量と処理時間の関係

8GBでも多重度の増加に合わせて性能向上が見られた．このように，既存バッチ処理をHadoopに適用すると，一定のサイズ以上で非常に効果的である．なお，従来のバッチ形式と同じ1多重の場合でも，大きく性能向上している．これは，Reduceタスクは1多重であるが，Mapタスクが入力ファイルの分割数だけ起動され，入力データの読み込みとシャッフル・ソートの一部処理を同時に行うことによって，高速化に寄与しているためである．

6. おわりに

本稿では，基幹バッチをHadoop上で実行するためのNetCOBOL Hadoop連携機能について述べた．基幹バッチにおいては，既存のアプリケーションとデータ資産が大量にあるため，これらを修正・変換することなく利用できることが重要である．特に，複数入力ファイルを用いる突合せを伴うアプリケーションはHadoopでの実行が困難なため，Hadoopマルチプレクサ技術を開発した．また，HadoopからCOBOL特有のデータフォーマットを扱うためのCOBOLデータ対応技術も開発した．

業務への実適用も行っており，アプリケーションの修正なく，並列実行することによる高速化の効果があることを示した．一方，当初想定していなかったデータの偏りによる並列性の限界がわかった．また異なるプラットフォームでデータを連携する際に処理時間がかかる点は，従来からあるデータマイグレーションと同様の課題であり今後の解決が待たれる．

参考文献

1) Hadoop, http://hadoop.apache.org/
2) FUJITSU Software NetCOBOL, 富士通 , http://software.fujitsu.com/jp/cobol/
3) Cloudera's Distribution Including Apache Hadoop, Cloudera, http://www.cloudera.com/content/cloudera/en/products/cdh.html
4) MapR Distribution for Apache Hadoop, MapR, http://www.mapr.com/products/
5) Hortonworks Data Platform, Hortonworks, http://hortonworks.com/products/hdp/
6) InfoSphere BigInsights, IBM, http://www-06.ibm.com/software/jp/data/infosphere/biginsights/
7) FUJITSU Software Interstage Big Data Parallel Processing Server, 富士通, http://interstage.fujitsu.com/jp/bigdatapps/
8) ノーチラス・テクノロジーズ，ノーチラス・テクノロジーズが株式会社アンデルセンサービスの原価計算の基幹バッチ処理を Asakusa Framework/Hadoop にて 1/12 の時間に短縮 (2012), http://www.nautilus-technologies.com/topics/20120507.html
9) Apache avro, http://avro.apache.org/
10) Apache thrift, http://thrift.apache.org/
11) COBOL 規格：http://www.cobol.gr.jp/knowledge/next_standard.html
12) Apache Hive, http://hive.apache.org/
13) Asakusa Framework, http://www.asakusafw.com/
14) Jimmy Lin and Chris Dyer: Hadoop MapReduce デザインパターン，オライリージャパン (2011).
15) Tom White，玉川竜司，兼田聖士：Hadoop 第 2 版，オライリー・ジャパン (2011).

上田 晴康（正会員）hal_ueda@jp.fujitsu.com
1989 年東京工業大学大学院理工学研究科情報科学専攻修士課程修了．同年富士通（株）入社．国際情報社会科学研究所（現，富士通研究所）に入所後，組合せ最適化および並列分散処理に関する研究に従事．人工知能学会会員．

榎田 勉（非会員）enokida.tsutomu@jp.fujitsu.com
1988 年静岡大学工学部情報工学科卒業．同年富士通（株）入社．現在，COBOL 開発・運用ソフトウェア「NetCOBOL」製品の開発に従事．

立岩 恭也（非会員）y-tateiwa@jp.fujitsu.com
2001 年中京大学情報科学部認知科学科卒業．同年（株）富士通インフォソフトテクノロジ入社．現在，富士通（株）勤務．NetCOBOL ランタイムの開発に従事．

採録決定：2013 年 12 月 27 日
編集担当：茂木　強（(独) 科学技術振興機構）